intel

APPLICATION NOTE

AP-81

January 1980

Inter-Intellec Communications

Herb Chin
Microcomputer Development Systems Applications

121558-001 Rev. A

RELATED INTEL DOCUMENTS

Peripheral Design Handbook, 9800676

Description of: 8251 Programmable Communications Interface
8253 Programmable Interval Timer

ISIS-II User's Guide, 9800306

Description of: ISIS-II System Calls

Intellec Series II Schematic Drawings, 9800554

Intellec Series II Installation & Service Manual, 9800559

Description of: Configuring the Serial Interfaces

ISIS-II Calls Application Note

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Inter-Intellec Communications

Table of Contents

I. INTRODUCTION

This application note discusses the implementation of a simple communications link between two Intellec Series II Development Systems. Such a communications link would allow files to be transmitted from one system to another, increasing the flexibility of program development on Intellec Systems.

The hardware and software aspects of constructing a communications link between two Intellec Series II Systems are described here in detail. An example communications link is also presented.

II. HARDWARE CONSIDERATIONS

The Intellec Series II has a variety of features to facilitate the establishment of a communications link. The existence of two (2) serial I/O channels allows the user the flexibility of having a serial device such as a printer, and having the ability to establish a serial communications link also.

These Serial I/O channels conform to EIA-RS232C specifications in the arrangement of the attention control lines. This allows a standard interface, between the Intellec System and a corresponding serial I/O device such as another Intellec Microcomputer Development System.

The serial I/O channels are driven by Intel 8251 programmable communications interface devices, also referred to as a USART (Universal Synchronous-Asynchronous Receiver-Transmitter). The 8251 USART is a very flexible device in that it can be programmed to operate in a variety of configurations and modes.

III. SOFTWARE CONSIDERATIONS

Interfacing an Intellec Development System to another development system through a serial I/O link requires a variety of software.

The 8251 USART is a programmable device and therefore requires specific attention in terms of the initialization of the environment in which the USART will operate. This initialization includes such things as selecting the baud rate and asynchronous or synchronous mode of operation.

The idea of transferring files requires the use of some sort of file handling software. The ISIS-II operating system allows flexible access to all user-files and therefore relieves the user of the burden of generating file I/O routines. This interface simply takes the form of ISIS-II system calls to open, close, read, and write to files.

The interface as seen by the user can also be facilitated by the use of ISIS-II system calls. The use of system input and output routines simplifies the retrieval of commands and the sending of status messages.

The code necessary to perform the actual transmission and reception of data can take on many forms, depending on the application. A possible algorithm will be discussed with the design example in the following sections.

IV. COMMUNICATIONS CONSIDERATIONS

The ability of the 8251 USART to operate in either asynchronous or synchronous mode raises the question of which mode is more appropriate for a particular application. The main difference is that the asynchronous mode requires framing information to be added to each character, while the synchronous format adds framing information to blocks of data. The synchronous format can transmit large blocks with less overhead but requires more complex decoding of the transmitted information. The asynchronous mode has higher overhead per character but alows for a simpler decoding environment.

As data is sent back and forth from one system to another, it requires some sort of organization. The use of some type of line protocol (a formal set of conventions governing the format of message exchange between two communicating processes), is necessary to facilitate the organization of what is being sent across the line, and in which a specific message will require a specific response from each computer. This will allow for a variety of circumstances to be handled. For example the handling of a data record that has not been transmitted correctly or where a data record must be remanipulated in some way to be correctly transmitted.

Special Communication Considerations

Utilizing the telephone network to connect two remote systems is an interesting application. This arrangement requires the use of modems for interfacing the Intellec Development Systems to the telephone network.

The Intellec Development System's serial channels have user-modifiable jumpers to allow easy hook-up of modems. The jumpers for direct hook-up are shown in our design example. Note that these differ from the jumpers used with a modem.

The use of acoustic couplers and the special problems that can be encountered will be discussed in the design example portion of this application note.

V. DESIGN EXAMPLE: SCOPE OF PROJECT

The example to be described in detail is a utility that allows a user to copy any type of ISIS-II file stored on disk from one Intellec Development System to another. The utility should require a minimum of hardware generation and limited changes to the existing Intellec System. The software should be able to transmit files and have the ability to verify and recover errors in transmission. The user interface should be simple and informative enough to satisfy the need for human interaction in the form of transmission verification and error recovery.

The following design example addresses the idea of having a direct hook-up between the two Intellec Development Systems. It also describes the idiosyncrasies and design decisions taken to meet the required specifications.

VI. DESIGN EXAMPLE: HARDWARE

Introduction

The only hardware modifications necessary for direct hook-up of two Intellec Development Systems is building a cable to interface the two systems, and coordinating the various attention control lines.

Details of Solution

The choice of which serial port to use is comparatively easy because the ISIS-II Operating System does not have a software driver for serial channel two (2). Serial channel one (1) has a driver and should be reserved for supporting various serial I/O devices (i.e., teletype terminals).

Serial channel two is the J3 connector on the connector strip of the IOC board (see figure 1). The lines on serial channel two conform to the specification for EIA-RS232C.

The lines that are necessary for direct hook-up are:

PINS – 1 Protective Ground
2 Transmitted Data
3 Received Data
4 Request to Send
5 Clear to Send
7 Signal Ground

The lines on pins 2 and 3 (transmitted-data and received data) must be crossed so that the transmit line of one port is linked to the receive line of the other port and vice versa on the receive line of the original system. This causes data being transmitted to go to a line that will receive data instead of another transmission line. This can be done most logically in the cable as shown in figure 2.

The lines on pins 4 and 5 (Request-To-Send and Clear-To-Send) must be tied together to establish that the USART are in a mode to receive and transmit data. This environment must be established as the 8251 USART uses these lines to acknowledge that it is in a mode to receive data and is through processing the previous data character. In this design the USART is always ready to accept data because the software driver processes the data character at a high enough rate that there is no loss of information.

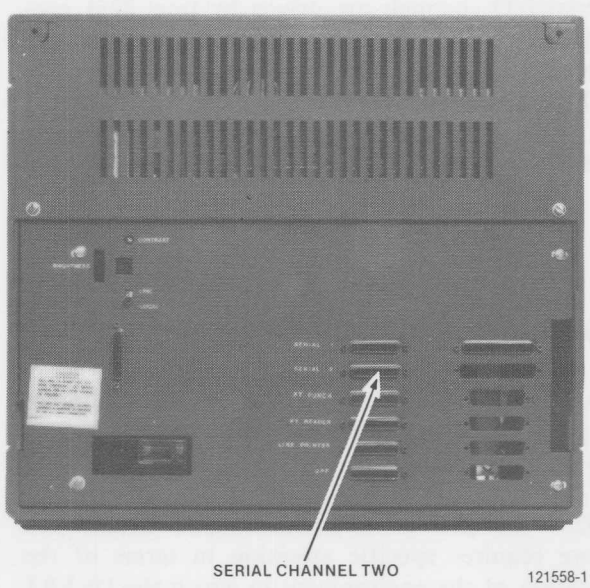
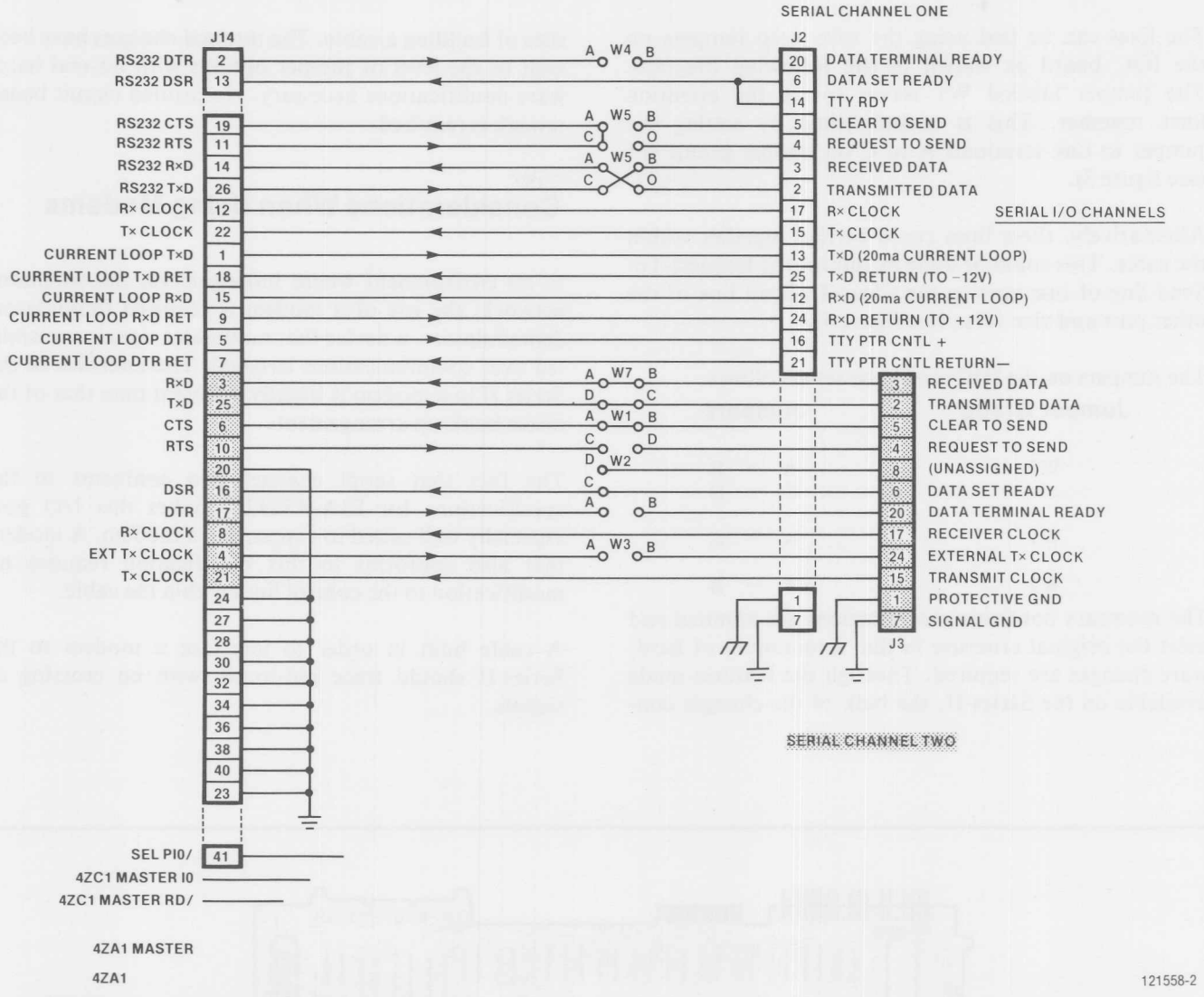


Figure 1. Rear View of Intellec® Series II



Intellec® Series II Serial Channels

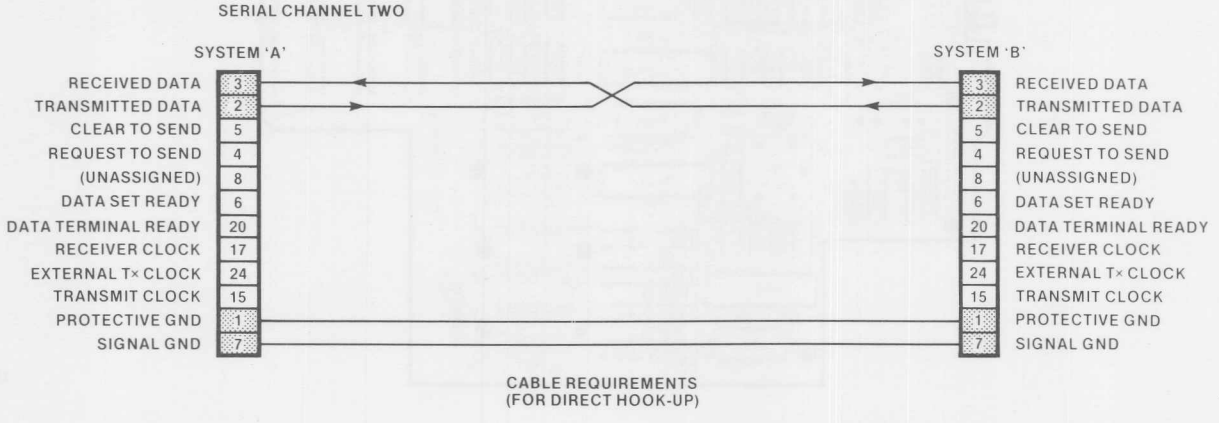


Figure 2

The lines can be tied using the wire-wrap jumpers on the IOC board as shown in the following diagram. The jumper labeled W1 serves to tie the attention lines together. This is accomplished by setting the jumper to link terminals A to C on jumper group W1 (see figure 3).

Alternatively, these lines could be tied together within the cable. This could be done by linking the Request-To-Send line of one port to the Clear-To-Send line of the other port and vice versa (see figure 4).

The jumpers on the IOC should be set as follows:

Jumper Group	Jumpers
W7	A – B C – D
W1	A – C
W2	A – B

The necessary hardware modifications are minimal and meet the original criterion in that a minimum of hardware changes are required. Through the features made available on the Series-II, the bulk of the changes consists of building a cable. The internal changes have been kept to the level of jumper options with no real hardware modifications necessary. No printed circuit board rework is required.

Considerations When Using Modems

In an environment where linking is via the telephone network, the use of a modem is required (modulator-demodulator—a device that modulates signals transmitted over communications circuits). The interface of the Series II to a modem is slightly different than that of the direct hook-up arrangement.

The fact that serial channel two conforms to the specifications for EIA-RS232C makes this I/O port especially well suited to supporting a modem. A modem that also conforms to this specification requires no modification to the control lines within the cable.

A cable built in order to interface a modem to the Series-II should trace pin-to-pin with no crossing of signals.

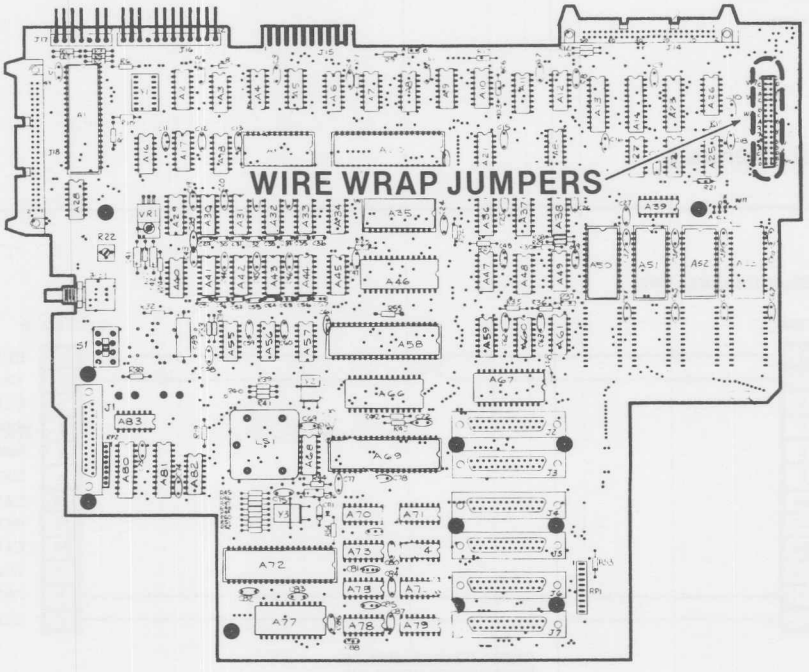


Figure 3. Input/Output Controller Board

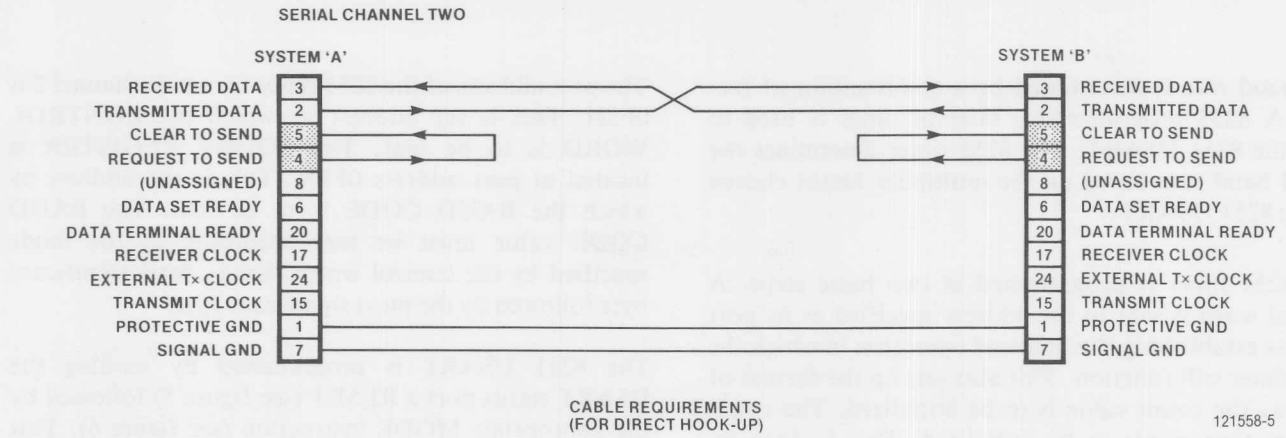


Figure 4

The jumpers on the Intellec System IOC should be set as follows:

Jumper Group	Jumpers
W7	A – B C – D
W1	A – B C – D
W2	A – B

VII. DESIGN EXAMPLE: SOFTWARE

Introduction

Linking two Intellec Development Systems is mainly a problem of the interaction between two communications programs.

Basically, two areas need to be addressed: the establishment of the execution environment, and the actual transfer of data to and from the communications routines. An additional consideration is the human interface between the communications routine and the user that invoked it. This includes the generation of informative error messages and indicators of data transfer assurance.

Setting the Serial Channel Environment

Establishing the environment begins with the initialization of the 8251 USART. The 8251 USART must be programmed into the desired mode of operation. The monitor program of the Series II initializes the USART at RESET time. In some cases, this initialization might not suit the desired mode of operation. For instance, the baud rate will not provide for efficient transfer of data.

In programming the 8251 USART a number of decisions must be made as to the operations mode. The format for transfer is to be asynchronous in this example, because it is a simpler environment in which to work. A choice must also be made among the following:

Baud Rate Factor	X1 X16 X64
Character Length	5 BITS 6 BITS 7 BITS 8 BITS
Parity Control	NO PARITY ODD PARITY EVEN PARITY
Framing Control	1 STOP BIT 1.5 STOP BITS 2 STOP BITS

The character length chosen is 8-bits because character data will be transferred between systems.

Parity control is odd in this case. This is arbitrary but must be consistent between the communicating systems.

Baud rate factor and framing control go together, because a certain number of stop bits is generally accepted for a certain rate.

Baud Rate	Number of Stop Bits
100	2
300	1
600	1
1200	1
2400	1
4800	1
9600	1

The baud rate will be chosen via a user-interactive medium with the appropriate framing control chosen by the communications routine.

The baud rate is determined by a combination of factors. A 8253 programmable interval timer is used to pace the 8251 USART. The 8253 timer determines the actual baud rate based on the multiplier factor chosen on the 8251 USART.

The 8253 timer is programmed in two basic steps. A control word is sent to the address specified as its port address establishing the mode of operation in which the 8253 timer will function. This also sets up the format of the way the count value is to be initialized. The count register then needs to be initialized. This is done by sending the appropriate BAUD CODE value to the count register port. The initialization of the 8253 timer determines the rate of the timing pulse used to set the pace at which the 8251 USART will function. This sets a range of possible baud rates via the multiplier factor of the 8251 USART. The following table shows the necessary combinations of 8253 timer BAUD CODES and the coordinating 8251 USART BAUD MULTIPLIER values.

Baud Rate	Baud Code	Baud Multiplier	X Factor
110	02BAH	0CEH	X16
300	0040H	04FH	X64
600	0020H	04FH	X64
1200	0010H	04FH	X64
2400	0020H	04EH	X16
4800	0010H	04EH	X16
9600	0000H	04EH	X16

8253 Timer Control Word

076H – Select Counter 1
Read/Load least significant byte first, then most significant
Square wave generator

The port address of the 8253 timer for serial channel 2 is 0F3H. This is the address by which the CONTROL WORD is to be sent. The COUNT REGISTER is located at port address 0F1H. This is the address by which the BAUD CODE is to be sent. The BAUD CODE value must be sent according to the mode specified by the control word; that is, least significant byte followed by the most significant byte.

The 8251 USART is programmed by sending the USART status port a RESET (see figure 5) followed by the appropriate MODE instruction (see figure 6). This MODE instruction is analogous to the value specified as the BAUD MULTIPLIER because this value contains the necessary information to set the other mode-related parameters such as parity and framing control. The next step in programming the 8251 USART is the sending of a COMMAND instruction (see figure 7). This step sets the actual operation of the selected format, such as enabling transmissions and/or reception of data.

The 8251 USART has two addresses that a user will manipulate: the status port and the data port. Serial channel two uses addresses 0F6H for the data port and 0F7H for the status port. The data port is the port where data will be read from or written to. The status port is the port where programming information and status of the serial line is read/written (see figure 8).

Serial Channel Two:

Device	Port Address
8253 Timer	0F1H – Count Register 0F3H – Control Port
8251 USART	0F6H – Data 0F7H – Status

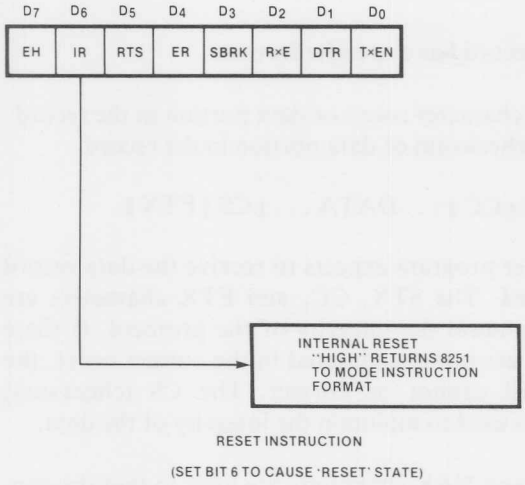


Figure 5

121558-6

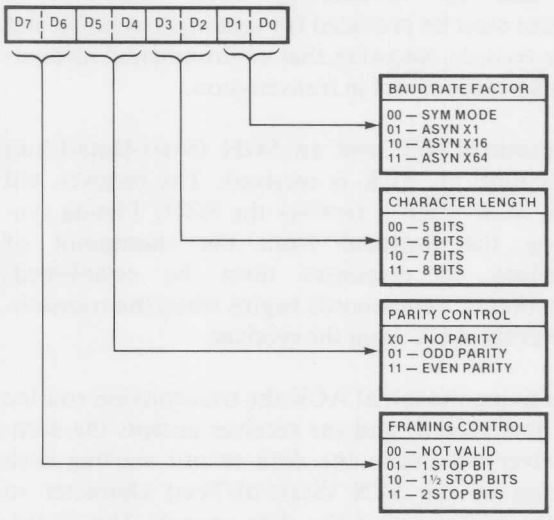


Figure 6

121558-7

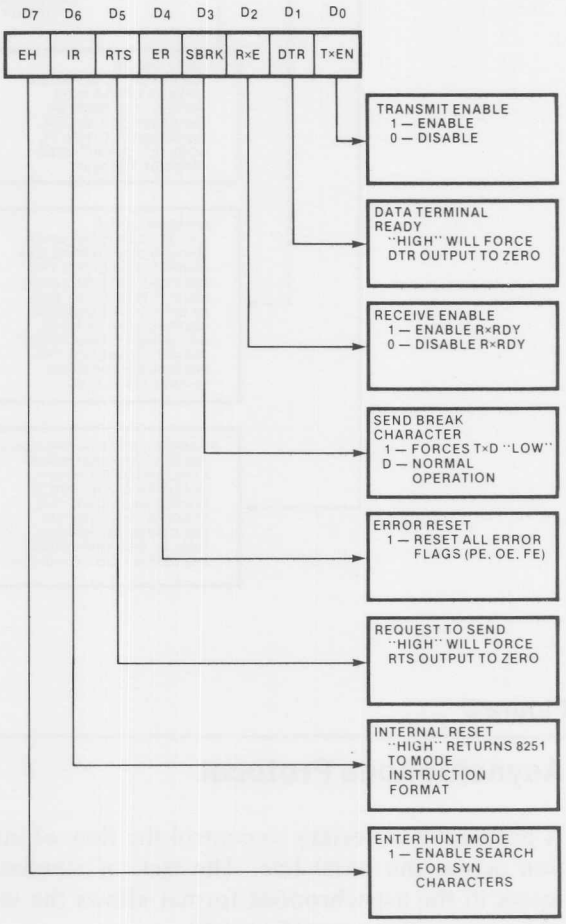


Figure 7

121558-8

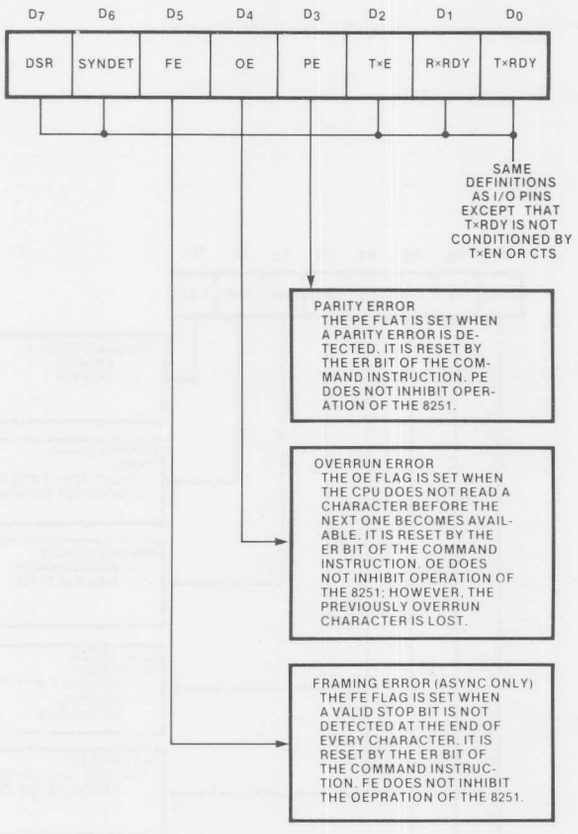


Figure 8 121558-9

Asynchronous Protocol

A protocol is necessary to control the flow of information across the serial line. The lack of standard protocols in the asynchronous format allows the tailoring of a protocol for a specific usage.

We chose a protocol where specific signals are sent from the transmitter and specific responses are required before proceeding to the next step. The following characters have been identified to represent certain messages:

Character	Meaning
SOH	Start Data Link
EOT	Terminate Data Link
STX	Start of Data Record
ETX	End of Data Record
ACK	Acknowledge
NAK	Negative Acknowledge
RS	Resend File – Described in Detail in Next Section

The SOH and EOT characters are used to establish the data link

The data record has a specific format:

CC = character count of data portion in the record
CS = checksum of data portion in the record

| STX | CC | . . . DATA . . . | CS | ETX |

The receiver program expects to receive the data record as described. The STX, CC, and ETX characters are used to maintain the integrity of the protocol. If these characters are not encountered in the correct order, the data record cannot be correct. The CS (checksum) character is used to maintain the integrity of the data.

The ACK and NAK characters are used to flag the condition of the data record. It is through the use of ACK and NAK that a handshaking scenario can exist between two communications programs.

Data Transfer Algorithm

The use of a protocol allows control of the information flow across the serial data line. This control is exercised by routines that will send and receive data. The use of a handshake arrangement allows for the handling of data records and for re-handling when necessary. A mechanism must be provided for sending/resending and receiving records, knowing that environmental idiosyncrasies may cause errors in transmission.

The transmitter will send an SOH (Start-Data-Link) character until an ACK is received. The receiver will send this ACK when it receives the SOH. Timing synchronizing the protocol from the standpoint of transmissions to responses must be established. Transmission of data records begins when the transmitter receives the ACK from the receiver.

After receiving the initial ACK the transmission routine sends a data record, and the receiver accepts the data. The receiver will parse the data record starting with verification of the STX (Start-of-Text) character to signify the beginning of the data record. This is followed by a character that has the character count of the actual data portion of the record. This character count is used to determine how much data is to be collected before the next control character. The character following the data represents a summation of the characters (checksum) calculated by the transmitter. The record is terminated by an ETX (End-of-Text) character signifying the end of this particular record.

The checksum value is calculated by the transmitter as the characters are being sent. A corresponding checksum is calculated by the receiver as characters are being read. When the transmitter's checksum is read, it is compared to the checksum generated by the receiver. If the two values are the same, an ACK is returned to the transmitter and another record is sent. If the two values do not agree, an error has been encountered, the data record is not valid, and a NAK is sent. The transmitter will repeat record if a NAK is received.

This scenario will continue until the last record is sent, at which time the transmitter will send a EOT signifying the end of the transmission.

Through the verification of the various protocol integrity values (i.e., STX) and the sequence of ACK/NAK with sending and resending of the data record, the integrity of the data transmission can be reasonably sure.

An additional integrity check is made by the parity bit which is supplied and checked by the 8251 USART. An error status can be read to verify integrity at this level of transmission. (See figure 8.)

Various parameters need to be considered in the data transmission phase. The possibility of error handling must be considered very seriously. The chance that a transmission error is unresolvable can occur. To prevent an infinite loop situation where one is constantly sending and resending, parameters for the number of retries to perform must be considered. The size of the data record also has an effect on the probability of error and the number of retries. The numbers chosen for the design example were:

DATA RECORD COUNT = 128 CHARACTERS
RETRY COUNT = 15

These numbers are purely arbitrary and under given circumstances, could either be higher or lower. The lower the character count, the higher the probability of an error occurrence for that record. This raises the overhead per character however, so tuning of these values should be determined depending on the execution environment.

The ISIS-II operating system provides mechanisms for accessing disk files. These mechanisms come in the form of various sytem calls to open, close, read, and write to these files. The use of these system calls completely frees the communications program from overhead requirements for file management tasks.

One way to handle files being transmitted or read is through the use of a memory buffer. This buffer holds the file during transmission and/or upon reception. Given a large file to transmit, the file is blocked into the buffer piece by piece. Instead of record by record manipulation, a buffer and pointers allows easy movement within the file.

This scenario for establishing the data link and subsequent transfer of data is one of many ways to do elementary inter-system communications. Less error checking may be necessary depending on the environment. A very large data record could also be considered. On the other hand given a very hostile environment more error checking may be necessary and a very small data record could be used.

Considerations When Using Modems

The use of modems to interface the telephone network presents a variety of circumstances that do not exist in the direct hook-up environment.

The main concern is the possibility of the phone line being terminated because of some mechanical break down in the phone system. Modems supply a carrier signal to signify that they are hooked-up together. If a carrier signal is confirmed by both modems, a status signal is supplied to a line on an RS232C interface. This is the Data-Set-Ready (DSR) line (PIN 6). The 8251 USART has a status bit in its status register that can be checked by a program to verify whether or not the modems are communicating.

Another consideration is the problem of the maximum transfer rate of the telephone line. When using acoustically coupled modems another constraint is presented, because most acoustically coupled modems operate at a maximum of 300 baud.

The use of modems allows the two development systems to exist anywhere there is a telephone. This increases the flexibility of the idea of file sharing.

User Interface

The user interface is a very important part of the usefulness of any utility.

This type of utility will inherently require the specification of source and destination files. The ISIS-II operating system allows for an easy method of fetching parameters. These parameters can be received from the console device by way of ISIS-II system calls oriented specifically for input and output to the console. Through the use of utilities, a user has an ISIS-II like interface. (A detailed description of these system calls can be found in the *ISIS-II User's Guide,* manual order number 9800306.

In the event that two development systems are widely separated it is possible that one system will be unable to respond to the other. Therefore, various time-out loops must be included at stategic locations of the transmit and receive loops. The amount of time one should wait is highly variable and should be based on the circumstances.

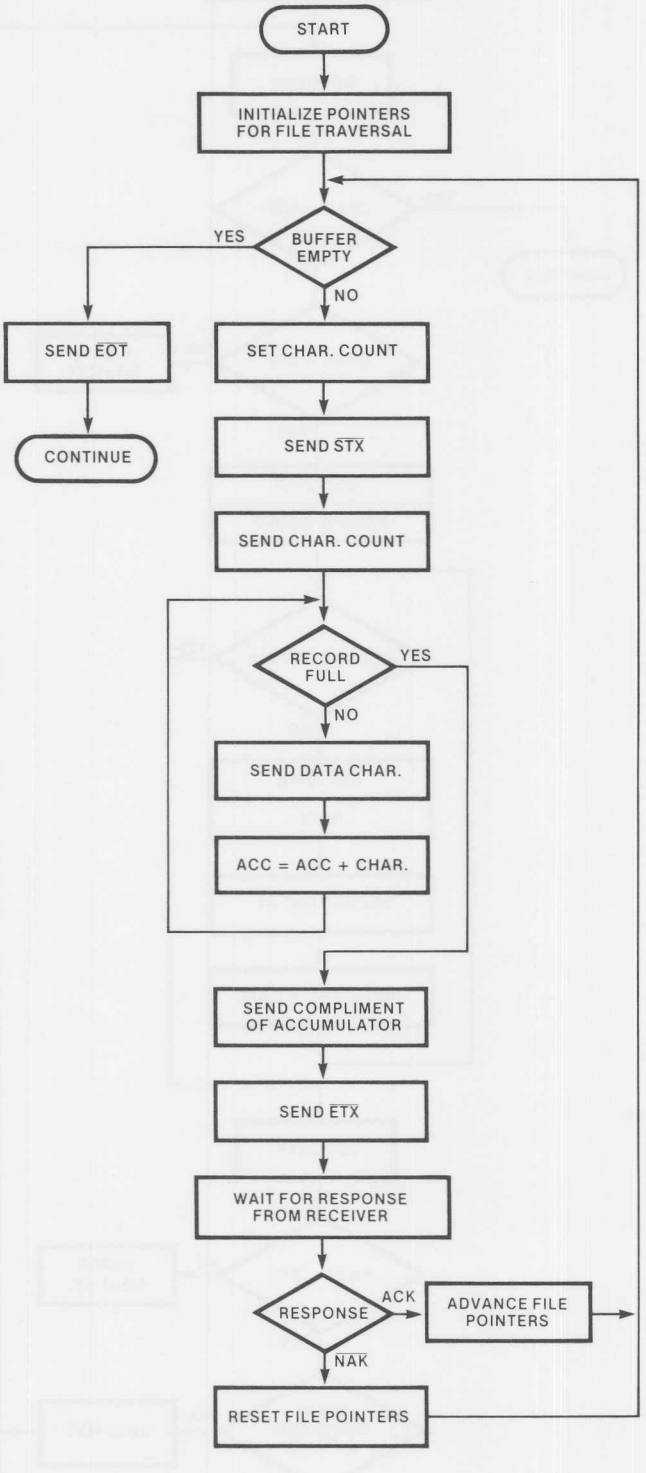
In this design example as part of the loop for transfer and receiving data, the two routines output a signal of confirmation to the console device.

VIII. CONCLUSION

This application note is intended to give an elementary view of inter-system communications. The example supplied is a simple and functional methodology for implementing a utility that will allow transfer of files from one Intellec Series II Development System to another.

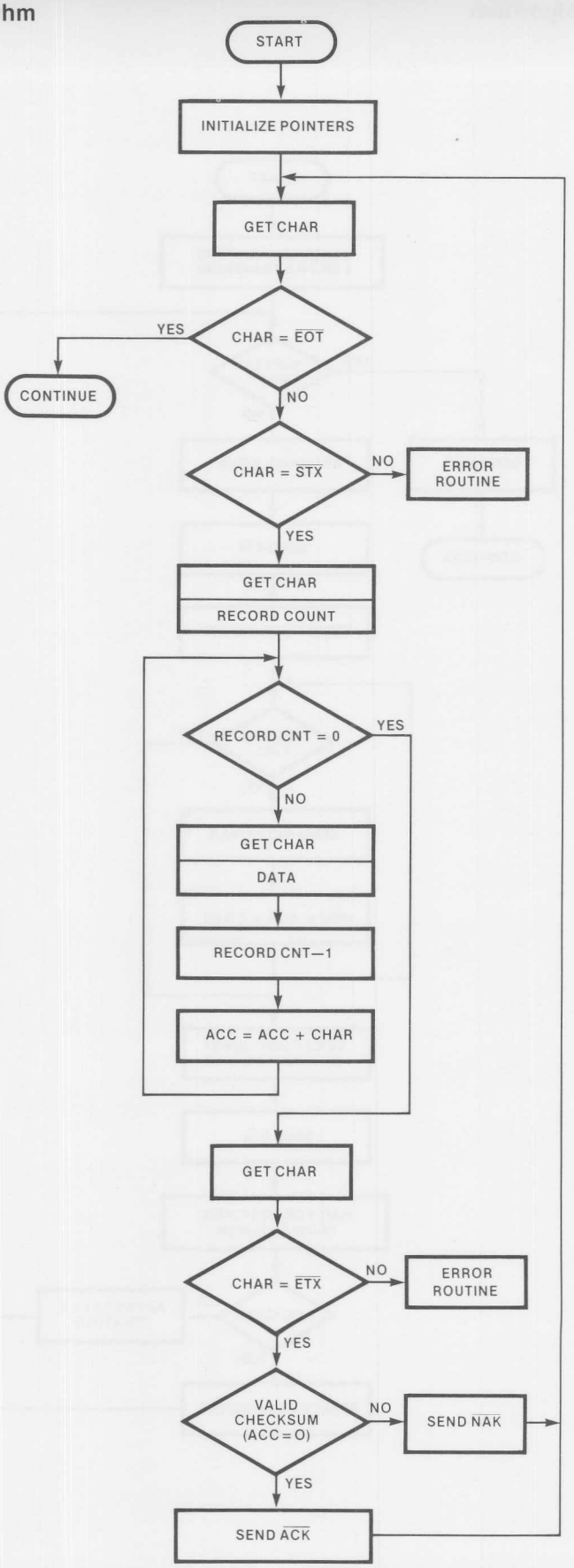
The program (design example) described here is available through the *Insite Users Library*.

Simplified Transmission Algorithm



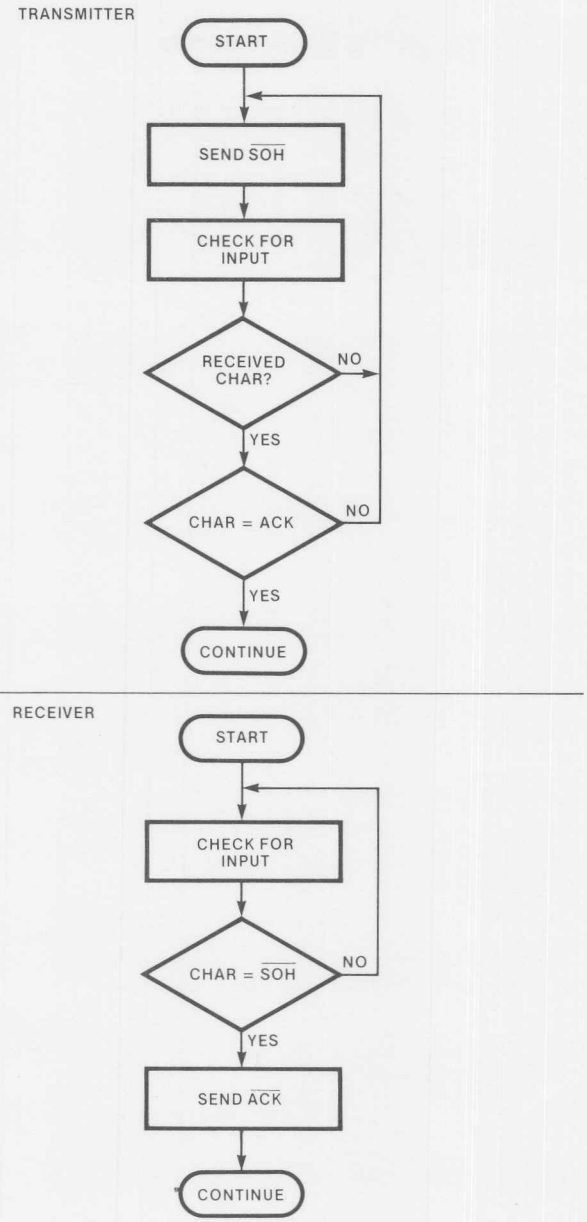
121558-10

Simplified Reception Algorithm



121558-11

Data-Link Algorithm



121558-12

APPENDIX A

CODING EXAMPLES

```
PL/M-80 COMPILER


ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SEND2
OBJECT MODULE PLACED IN :F1:SEND2.OBJ
COMPILER INVOKED BY:  PLM80 :F1:SEND2.PLM DEBUG


   1            SEND2:  DO;
                /*****************************************************************/

   2    1       PUTSCHAR: PROCEDURE(CHAR) EXTERNAL;
   3    2           DECLARE CHAR BYTE;
   4    2       END PUTSCHAR;

   5    1       GETSCHAR: PROCEDURE BYTE EXTERNAL;
   6    2       END GETSCHAR;

   7    1       STARTSUPSMESSAGE: PROCEDURE EXTERNAL;
   8    2       END STARTSUPSMESSAGE;

   9    1       TRANSMITSMESSAGE: PROCEDURE (INDEX) EXTERNAL;
  10    2            DECLARE INDEX BYTE;
  11    2       END TRANSMITSMESSAGE;

  12    1       TRXNET: PROCEDURE BYTE EXTERNAL;
  13    2       END TRXNET;

  14    1       DECLARE VERSFLAG  BYTE EXTERNAL;

                /*****************************************************************/

  15    1       XMIT:   PROCEDURE(FNAMESPTR,BUFSPTR,FCOUNT,BCOUNT,ERR) PUBLIC;

                /* THIS PROCEDURE TRANSMITS THE FILE RECORD BY RECORD (128 CHARS) */
                /* ACROSS A COMMUNICATIONS LINE                                   */
                /*                                                                */
                /*            EXIT MODES ARE AS FOLLOWS:                          */
                /*               ERROR = 0    :=    NORMAL RETURN                 */
                /*               ERROR = 1    :=    ABORT: MODEM OFF              */
                /*               ERROR = 2    :=    TERM: ABNORMAL TRANSMISSION   */
                /*               ERROR = 3    :=    NORES: NO CHARS RECEIVED      */
                /*               ERROR = 4    :=    TIMOUT: UNABLE TO SEND CHARS  */
                /*               ERROR = 5    :=    BADT: ERROR IN TRANSMISSION   */
```

```
PL/M-80 COMPILER

                $EJECT
  16    2           DECLARE FNAMESPTR       ADDRESS;
  17    2           DECLARE BUFSPTR         ADDRESS;
  18    2           DECLARE FCOUNT          ADDRESS;
  19    2           DECLARE BCOUNT          ADDRESS;
  20    2           DECLARE ERR             ADDRESS;

  21    2           DECLARE FILENAME        BASED FNAMESPTR (122) BYTE;
  22    2           DECLARE BUFFER          BASED BUFSPTR (35000) BYTE;
  23    2           DECLARE ERROR           BASED ERR             ADDRESS;

  24    2           DECLARE FILESPTR        ADDRESS;
  25    2           DECLARE RETRYSPTR       ADDRESS;
  26    2           DECLARE ATTNSIN         ADDRESS;
  27    2           DECLARE RECSCNT         ADDRESS;
  28    2           DECLARE I               ADDRESS;

  29    2           DECLARE J               BYTE;

  30    2           DECLARE CHKSACC         BYTE;
  31    2           DECLARE SEMAPHORE       BYTE;

  32    2           DECLARE STX             BYTE DATA(02H);
  33    2           DECLARE ETX             BYTE DATA(03H);
  34    2           DECLARE EOT             BYTE DATA(04H);
  35    2           DECLARE RS              BYTE DATA(1EH);
  36    2           DECLARE NAK             BYTE DATA(15H);
  37    2           DECLARE ACK             BYTE DATA(06H);
  38    2           DECLARE YES             BYTE DATA(0FFH);
```

```
PL/M-80 COMPILER

                $EJECT
                /* CHECK STATUS OF COMMUNICATIONS LINE */
  39    2           DO CASE TRXNET;
  40    3              ;
  41    3              GO TO ABORT;
  42    3              GO TO TERMINATE;
  43    3              GO TO NOSRESPONSE;
  44    3              GO TO TIMEOUT;
  45    3           END;

                /* INITIALIZE CHECKSUM ACCUMULATOR */
  46    2       FLOOP:  CHKSACC = 0;

                /* SEND FILENAME RECORD PRECEEDED BY START OF */
                /*                      TRANSMISSION CHARACTER   */
  47    2           CALL PUTSCHAR(STX);
  48    2           CALL PUTSCHAR(FCOUNT+2);
  49    2           DO I = 0 TO FCOUNT;
  50    3              CALL PUTSCHAR(FILENAME(I));
  51    3              CHKSACC = CHKSACC + FILENAME(I);
  52    3           END;
                /* SEND CHECKSUM */
  53    2           CALL PUTSCHAR(-CHKSACC);
                /* SEND END OF TRANSMISSION CHARACTER */
  54    2           CALL PUTSCHAR(ETX);

                /* WAIT FOR RESPONSE FROM RECEIVER */
                /* CHECK FOR VALIDITY OF TRANSMISSION */
  55    2           SEMAPHORE = GETCHAR;
  56    2           IF SEMAPHORE = NAK THEN GO TO FLOOP;
  58    2           IF SEMAPHORE <> ACK THEN GO TO BADTRANS;

                /* SEND THE 'START-UP' MESSAGE TO THE :CO:*/
  60    2           CALL STARTSUPSMESSAGE;

                /* SET UP POINTERS FOR FILE TRAVERSAL */
  61    2           FILESPTR = 0;
  62    2           RETRYSPTR = 0;
  63    2           J = 0;

                /* SEND FILE RECORD BY RECORD WITH RETRIES */
  64    2           DO WHILE (BCOUNT <> FILESPTR);
                /* SEND A BYTE-COUNT */
  65    3              RECSCNT = BCOUNT - FILESPTR;
  66    3              IF RECSCNT > 128 THEN
  67    3                 RECSCNT = 128;
  68    3              CHKSACC = 0;
  69    3              CALL PUTSCHAR(STX);
  70    3              CALL PUTSCHAR(RECSCNT+1);
  71    3              DO WHILE (RECSCNT <> 0) AND (BCOUNT <> FILESPTR);
  72    4                 CALL PUTSCHAR(BUFFER(FILESPTR));
  73    4                 CHKSACC = CHKSACC + BUFFER(FILESPTR);
  74    4                 FILESPTR = FILESPTR + 1;
  75    4                 RECSCNT = RECSCNT - 1;
  76    4              END;
  77    3              CALL PUTSCHAR(-CHKSACC);
  78    3              CALL PUTSCHAR(ETX);

PL/M-80 COMPILER
```

```
PL/M-80 COMPILER                                                    PAGE   4


                $EJECT
                /* CHECK FOR VALIDITY OF TRANSMISSION */
  79    3               SEMAPHORE = GETCHAR;
  80    3               IF SEMAPHORE = ACK THEN RETRYSPTR = FILESPTR;
  82    3                  ELSE IF SEMAPHORE = NAK THEN FILESPTR = RETRYSPTR;
  84    3                     ELSE GO TO BADTRANS;
  85    3                  CALL TRANSMITSMESSAGE(J);
  86    3                  J = J + 1;
  87    3                  IF J > 11 THEN
  88    3                     J = 0;
  89    3           END;


                /* SEND 'END OF TRANSMISSION' CHARACTER */
  90    2           IF VERSFLAG = YES THEN
  91    2              CALL PUTCHAR(RS);
                    ELSE
  92    2              CALL PUTSCHAR(EOT);

                /* EXIT MODES */
  93    2           NORMAL: ERROR = 0;
  94    2               RETURN;

  95    2           ABORT: ERROR = 1;
  96    2               RETURN;

  97    2           TERMINATE: ERROR = 2;
  98    2               RETURN;

  99    2           NOSRESPONSE: ERROR = 3;
 100    2               RETURN;

 101    2           TIMESOUT: ERROR = 4;
 102    2               RETURN;

 103    2           BADTRANS: ERROR = 5;
 104    2               RETURN;

 105    2           END XMIT;

 106    1           END SEND2;


MODULE INFORMATION:

     CODE AREA SIZE     = 0211H     529D
     VARIABLE AREA SIZE = 0017H      23D
     MAXIMUM STACK SIZE = 0006H       6D
     163 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER                                                    PAGE   1


ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE RECV2
OBJECT MODULE PLACED IN :F1:RECV2.OBJ
COMPILER INVOKED BY:  PLM80 :F1:RECV2.PLM DEBUG


   1            RECV2: DO;
                /*****************************************************************/

   2    1       DECLARE VERSFLAG BYTE EXTERNAL;

   3    1       CLOSE: PROCEDURE(AFTNPTR,STATUS) EXTERNAL;
   4    2           DECLARE (AFTNPTR,STATUS) ADDRESS;
   5    2       END CLOSE;

   6    1       GETSCHAR: PROCEDURE BYTE EXTERNAL;
   7    2       END GETSCHAR;

   8    1       OPEN:  PROCEDURE(AFTNPTR,FILE,ACCESS,MODE,STATUS) EXTERNAL;
   9    2           DECLARE (AFTNPTR,FILE,ACCESS,MODE,STATUS) ADDRESS;
  10    2       END OPEN;

  11    1       PUTSCHAR: PROCEDURE(CHAR) EXTERNAL;
  12    2           DECLARE CHAR      BYTE;
  13    2       END PUTSCHAR;

  14    1       STARTSUPSMESSAGE: PROCEDURE EXTERNAL;
  15    2       END STARTSUPSMESSAGE;

  16    1       TRANSMITSMESSAGE: PROCEDURE (INDEX) EXTERNAL;
  17    2           DECLARE INDEX BYTE;
  18    2       END TRANSMITSMESSAGE;

  19    1       WRITE: PROCEDURE(AFTNPTR,BUFFER,COUNT,STATUS) EXTERNAL;
  20    2           DECLARE (AFTNPTR,BUFFER,COUNT,STATUS) ADDRESS;
  21    2       END WRITE;

                /*****************************************************************/

  22    1       RCVR: PROCEDURE (FNAMEPTR,BUFPTR,ERR) PUBLIC;

                /* THIS PROCEDURE PERFORMS THE FUNCTION OF BUILDING AN IN MEMORY  */
                /* BUFFER FROM COMMUNICATIONS LINE INPUT AND THEN WRITES THIS     */
                /* BUFFER TO A DISKETTE FILE DEFINED BY THE TRANSMITTING ROUTINE. */
                /*                                                                */
                /*        EXIT MODES ARE AS FOLLOWS:                              */
                /*              ERROR = 0   :=   NORMAL EXIT                      */
                /*              ERROR = 1   :=   ABNORMAL TERMINATION             */
                /*              ERROR = 2   :=   ERROR IN TRANSMISSION            */
```

```
                $EJECT
 23    2          DECLARE NUMSRETRY          LITERALLY '15';
 24    2          DECLARE USART2$STATUS$PORT LITERALLY '0F7H';
 25    2          DECLARE ERRSMASK           LITERALLY '38H';

 26    2          DECLARE FNAMESPTR          ADDRESS;
 27    2          DECLARE BUFSPTR            ADDRESS;
 28    2          DECLARE ERR                ADDRESS;

 29    2          DECLARE FILENAME           BASED FNAMESPTR (122) BYTE;
 30    2          DECLARE BUFFER             BASED BUFSPTR (35000) BYTE;
 31    2          DECLARE ERROR              BASED ERR          ADDRESS;

 32    2          DECLARE FILESPTR           ADDRESS;
 33    2          DECLARE RETRYSPTR          ADDRESS;
 34    2          DECLARE AFTNSOUT           ADDRESS;

 35    2          DECLARE RETRY              BYTE;
 36    2          DECLARE CHKSACC            BYTE;
 37    2          DECLARE CHAR               BYTE;
 38    2          DECLARE RECSCNT            BYTE;
 39    2          DECLARE BADSCHARSFLAG      BYTE;
 40    2          DECLARE ERRORSFLAG         BYTE;

 41    2          DECLARE J                  BYTE;

 42    2          DECLARE STX                BYTE DATA(02H);
 43    2          DECLARE ETX                BYTE DATA(03H);
 44    2          DECLARE EOT                BYTE DATA(04H);
 45    2          DECLARE RS                 BYTE DATA(1EH);
 46    2          DECLARE ACK                BYTE DATA(06H);
 47    2          DECLARE NAK                BYTE DATA(15H);
 48    2          DECLARE YES                BYTE DATA(0FFH);
 49    2          DECLARE NO                 BYTE DATA(00H);
```

```
                $EJECT
                /* OPEN DESTINATION FILE AS SPECIFIED BY CALLING ROUTINE */
 50    2            CALL OPEN(.AFTN$OUT,.FILENAME,2,0,.ERROR);
 51    2            IF ERROR <> 0 THEN GO TO TERMINATE;

                /* INITIALIZE FILE TRAVERSAL POINTERS */
 53    2            FILE$PTR = 0;
 54    2            RETRY$PTR = 0;
 55    2            RETRY = NUM$RETRY;
 56    2            CHK$ACC = 0;
 57    2            BAD$CHARS$FLAG = 0;
 58    2            ERROR$FLAG = 0;
 59    2            J = 0;

                /* SEND A 'COMMUNICATIONS BEGUN' MESSAGE TO THE :CO: */
 60    2            CALL START$UP$MESSAGE;

                /* SEND AN ACKNOWLEDGE FOR THE FILENAME RECORD */
 61    2            CALL PUT$CHAR(ACK);

                /* SEND FOR THE STX CHARACTER */
 62    2            CHAR = GET$CHAR;

                /* BUILDS MEMORY BUFFER WITH COMMUNICATIONS INPUT */
                /* UNTIL END OF TRANSMISSION CHARACTER IS READ    */
 63    2            DO WHILE (CHAR <> EOT) AND (CHAR <> RS);
 64    3               IF CHAR <> STX THEN BAD$CHARS$FLAG = 0FFH;
 66    3               REC$CNT = GET$CHAR;
 67    3               CHAR = GET$CHAR;
 68    3               DO WHILE (REC$CNT <> 0);
 69    4                  BUFFER(FILE$PTR) = CHAR;
 70    4                  FILE$PTR = FILE$PTR + 1;
 71    4                  REC$CNT = REC$CNT - 1;
 72    4                  CHK$ACC = CHK$ACC + CHAR;
 73    4                  CHAR = GET$CHAR;
 74    4               END;
 75    3               IF CHAR <> ETX THEN BAD$CHARS$FLAG = 0FFH;
 77    3               IF (INPUT(USART2$STATUS$PORT) AND ERR$MASK) <> 0 THEN
 78    3                  DO;
 79    4                     ERROR$FLAG = 0FFH;
 80    4                     OUTPUT(USART2$STATUS$PORT) = 37H;
 81    4                  END;

                /* TAKE CARE OF CHECKSUM IN BUFFER */
 82    3               FILE$PTR = FILE$PTR - 1;

 83    3               CALL TRANSMIT$MESSAGE(J);
 84    3               J = J + 1;
 85    3               IF J > 11 THEN
 86    3                  J = 0;

 87    3               IF (CHK$ACC <> 0) OR (BAD$CHARS$FLAG <> 0) OR (ERROR$FLAG <> 0) THEN
 88    3                  DO;
 89    4                    FILE$PTR = RETRY$PTR;
 90    4                    RETRY = RETRY - 1;
 91    4                    IF RETRY = 0 THEN GO TO BADTRANS;
 93    4                    CALL PUT$CHAR(NAK);
```

```
PL/M-80 COMPILER                                                    PAGE   4


  94    4                   END;
  95    3                   ELSE DO;
  96    4                          RETRYSPTR = FILESPTR;
  97    4                          RETRY = NUMSRETRY;
  98    4                          CALL PUTCHAR(ACK);
  99    4                        END;
 100    3                CHKSACC = 0;
 101    3                BADSCHARSFLAG = 0;
 102    3                ERRORSFLAG =0;
 103    3                CHAR = GETSCHAR;
 104    3            END;

 105    2            IF CHAR = RS THEN
 106    2               VERSFLAG = YES;
                      ELSE
 107    2               VERSFLAG = NO;

                    /* WRITE COMPLETE BUFFER TO DISK */
 108    2            CALL WRITE(AFTNSOUT,.BUFFER,FILESPTR,.ERROR);
 109    2            IF ERROR <> 0 THEN GO TO TERMINATE;

 111    2            CALL CLOSE(AFTNSOUT,.ERROR);
 112    2            IF ERROR <> 0 THEN GO TO TERMINATE;

                    /* EXIT MODES */

 114    2          NORMAL: ERROR = 0;
 115    2            RETURN;

 116    2          TERMINATE: ERROR = 1;
 117    2            RETURN;

 118    2          BADTRANS: ERROR = 2;
 119    2            RETURN;

 120    2          END RCVR;

 121    1          END RECV2;


MODULE INFORMATION:

     CODE AREA SIZE     = 01EAH     490D
     VARIABLE AREA SIZE = 0013H      19D
     MAXIMUM STACK SIZE = 0008H       8D
     172 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER     CHANGE BAUD RATE FOR SERIES-II                    PAGE   1


ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE BAUDRT
OBJECT MODULE PLACED IN :F1:BAUDRT.OBJ
COMPILER INVOKED BY:  PLM80 :F1:BAUDRT.PLM


             $PAGEWIDTH(80) DEBUG TITLE('CHANGE BAUD RATE FOR SERIES-II')

   1         BAUDRT: DO;
                        /* THIS MODULE CONTAINS THE PROCEDURE 'BAUD$RATE'
                        THAT CHANGES THE BAUD RATE OF SERIAL CHANNEL 1
                        ON A SERIES-II DEVELOPMENT SYSTEM AND CHAREOL FOR
                        COMPARING 2 BYTE STRINGS.  IT ALSO CONTAINS
                        STATCHK TO CHECK FOR ISIS ERRORS. */

   2    1              WRITE: PROCEDURE (AFT, BUF, CNT, STAT) EXTERNAL;
   3    2                       DECLARE (AFT, BUF, CNT, STAT) ADDRESS;
   4    2              END WRITE;

   5    1              READ: PROCEDURE (AFT, BUF, CNT, ACT, STAT) EXTERNAL;
   6    2                      DECLARE (AFT, BUF, CNT, ACT, STAT) ADDRESS;
   7    2              END READ;

   8    1              ERROR: PROCEDURE (ERR) EXTERNAL;
   9    2                       DECLARE  ERR  ADDRESS;
  10    2              END ERROR;

  11    1              EXIT: PROCEDURE EXTERNAL;
  12    2              END EXIT;
```

```
                $EJECT

 13   1        STATCHK: PROCEDURE (ERRNUM) PUBLIC;

                        /* THIS PROCEDURE CHECKS FOR AN
                        ISIS ERROR, THEN PRINTS IT AND
                        RETURNS TO ISIS IF THER IS ONE. */

 14   2          DECLARE ERRNUM ADDRESS;

 15   2          IF ERRNUM > 0 THEN
 16   2          DO;
 17   3            CALL ERROR (ERRNUM);
 18   3            CALL EXIT;
 19   3          END;

 20   2        END STATCHK;
```

```
                $EJECT

 21    1        CHAREOL: PROCEDURE (STRG1PTR, STRG2PTR, LEN) BYTE PUBLIC;

                        /* COMPARISON OF TWO BYTE STRINGS.
                        RETURN 1 IF =.
                        RETURN 0 IF STRG1 < STRG2,
                        RETURN 2 IF STRG1 > STRG2 */

 22    2        DECLARE
                    (STRG1PTR, STRG2PTR) ADDRESS,
                    STRG1 BASED STRG1PTR (1) BYTE,
                    STRG2 BASED STRG2PTR (1) BYTE,
                    LEN BYTE,
                    I BYTE;

 23    2        I = 0;
 24    2        DO WHILE (STRG1(I) = STRG2(I) AND I < LEN);
 25    3          I = I + 1;
 26    3        END;

 27    2        IF I = LEN THEN RETURN 1;
 29    2        IF STRG1(I) < STRG2(I) THEN RETURN 0;
 31    2        RETURN 2;

 32    2        END CHAREOL;
```

```
                $EJECT

 33    1        BAUD$RATE: PROCEDURE PUBLIC;

                          /* THIS PROCEDURE READS IN THE DESIRED
                          BAUD RATE FROM THE CONSOLE & THEN
                          CHANGES THE RATE.  */

 34    2            DECLARE
                        VALID$BAUD$RATES (*) BYTE DATA
                            ('110', 0DH,
                             '150', 0DH,
                             '300', 0DH,
                             '600', 0DH,
                             '1200',
                             '2400',
                             '4800',
                             '9600'),

                        BAUD$CODES (*) ADDRESS DATA
                            (2BAH,
                              80H,
                              40H,
                              20H,
                              10H,
                              20H,
                              10H,
                              08H),

                        BAUD$MULTS (*) BYTE DATA
                            (0CEH,
                              4FH,
                              4FH,
                              4FH,
                              4FH,
                              4FH,
                              4FH,
                              4FH);

 35    2            DECLARE
                        BAUD$PROMPT (*) BYTE DATA
                            (0DH, 0AH, 0AH, 'ENTER BAUD RATE FOR SERIAL CHANNEL',
                            ' 1 (0 FOR LIST OF RATES) :'),

                        END$MSG (*) BYTE DATA
                            (0DH, 0AH, 'BAUD RATE INITIALIZATION COMPLETED.',
                            0DH,0AH);

 36    2            DECLARE
                        (I,BAUD$RATE$INDEX) BYTE,
                        (STATUS, ACTUAL) ADDRESS,
                        BAUD$INPUT (8) BYTE;
```

```
$EJECT

37  2       DECLARE
                AS LITERALLY 'LITERALLY',

                CNTLSPORT$8253 AS '0F3H',
                COUNTSREG$8253 AS '0F1H',
                CMDSPORT$8251  AS '0F7H',

                CONTROLSBYTE$8253   AS '76H',
                RESET$8251$TO$MODE  AS '40H',
                ENABLE$8251         AS '27H',

                INVALID$BAUD$RATE   AS '7FH';
```

```
                $EJECT

 38   2         PRINTSBAUDSRATES: PROCEDURE;

                        /* THIS PROCEDURE PRINTS A LIST OF
                        VALID BAUD RATES ON THE CONSOLE.  IT
                        DOES NOT NEED TO BE MODIFIED IF THE
                        NUMBER OR VALUES OF VALID BAUD RATES
                        CHANGE.  */

 39   3         DECLARE
                    RATESMSG (*) BYTE DATA
                      (0DH, 0AH, 'VALID BAUD RATES ARE '),
                    LEN BYTE,
                    LIST$BUF (10) BYTE;

 40   3         CALL WRITE (0, .RATESMSG, LENGTH(RATE$MSG), .STATUS);
 41   3         CALL STATCHK (STATUS);

 42   3         DO I = 0 TO LENGTH(VALIDSBAUDSRATES)-4 BY 4;

 43   4           IF VALIDSBAUDSRATES(I+3) = 0DH THEN LEN = 5;
 45   4           ELSE LEN = 6;

 46   4           CALL MOVE (LEN-2, .VALIDSBAUDSRATES(I), .LIST$BUF);

 47   4           IF I = LENGTH(VALIDSBAUDSRATES)-4 THEN
 48   4           DO;
 49   5             LIST$BUF (LEN-2) = '.';
 50   5             LEN = LEN - 1;
 51   5           END;
 52   4           ELSE DO;
 53   5             LIST$BUF (LEN-2) = ',';
 54   5             LIST$BUF (LEN-1) = ' ';
 55   5           END;

 56   4           CALL WRITE (0, .LIST$BUF, LEN, .STATUS);
 57   4           CALL STATCHK (STATUS);

 58   4         END;
 59   3         END PRINTSBAUDSRATES;
```

```
PL/M-80 COMPILER    CHANGE BAUD RATE FOR SERIES-II                    PAGE   7


             $EJECT

                          /* READ IN BAUD RATE FROM OPERATOR
                          AT CONSOLE */

  60    2           BAUD$RATE$INDEX = INVALID$BAUD$RATE;

  61    2           DO WHILE BAUD$RATE$INDEX = INVALID$BAUD$RATE;

  62    3             CALL WRITE (0, .BAUD$PROMPT, LENGTH(BAUD$PROMPT), .STATUS);
  63    3             CALL STATCHK (STATUS);

  64    3             CALL READ (1, .BAUD$INPUT, 20, .ACTUAL, .STATUS);
  65    3             CALL STATCHK (STATUS);

  66    3             DO I = 0 TO LENGTH(VALID$BAUD$RATES)-4 BY 4;
  67    4               IF CHAREQL (.BAUD$INPUT, .VALID$BAUD$RATES(I), 4) THEN
  68    4                 BAUD$RATE$INDEX = I / 4;
  69    4             END;

  70    3             IF BAUD$RATE$INDEX = INVALID$BAUD$RATE THEN
  71    3               CALL PRINT$BAUD$RATES;

  72    3           END;

                          /* OUTPUT TO INTERVAL TIMER (8253) AND
                          USART (8251) THE PROPER COMMANDS  */

  73    2           OUTPUT (CNTL$PORT$8253) = CONTROL$BYTE$8253;
  74    2           OUTPUT (COUNT$REG$8253) = LOW (BAUD$CODES(BAUD$RATE$INDEX));
  75    2           OUTPUT (COUNT$REG$8253) = HIGH(BAUD$CODES(BAUD$RATE$INDEX));

  76    2           OUTPUT (CMD$PORT$8251) = RESET$8251$TO$MODE;
  77    2           OUTPUT (CMD$PORT$8251) = BAUD$MULTS (BAUD$RATE$INDEX);
  78    2           OUTPUT (CMD$PORT$8251) = ENABLE$8251;

  79    2           CALL WRITE (0, .END$MSG, LENGTH(END$MSG), .STATUS);
  80    2           CALL STATCHK (STATUS);

  81    2           RETURN;

  82    2       END BAUD$RATE;

  83    1       END BAUDRT;


MODULE INFORMATION:

     CODE AREA SIZE     = 02F8H     760D
     VARIABLE AREA SIZE = 0021H      33D
     MAXIMUM STACK SIZE = 0008H       8D
     217 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```